Ex.1

```
# Import image
lena = cv2.imread('sift_images/lena.bmp')
# Convert to gray-scale
lena_gray = cv2.cvtColor(lena, cv2.COLOR_BGR2GRAY)

# Define kernels
x_filter = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
y_filter = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
# Compute G_x and G_y
grad_x = cv2.filter2D(lena_gray, cv2.CV_64F, x_filter)
grad_y = cv2.filter2D(lena_gray, cv2.CV_64F, y_filter)
# Compute G_x and G_y (using Sobel operator)
grad_x_sob = cv2.Sobel(lena_gray, cv2.CV_64F, 1, 0)
grad_y_sob = cv2.Sobel(lena_gray, cv2.CV_64F, 0, 1)
# Check differences
diff_x = grad_x - grad_x_sob
diff_y = grad_y - grad_y_sob
print(f"G_x (filter2D) - G_x (Sobel): {diff_x.sum()}")
print(f"G_y (filter2D) - G_y (Sobel): {diff_y.sum()}")
# Absolute magnitude
mag = np.sqrt(grad_x**2 + grad_y**2)

# Convert images to 8-bit
grad_x = cv2.convertScaleAbs(grad_x)
grad_y = cv2.convertScaleAbs(grad_y)
mag = cv2.convertScaleAbs(mag)

plt.figure(figsize=(20,25))
_ = plt.subplot(1,4,1), plt.imshow(lena_gray, cmap=plt.cm.gray), plt.title('Input image')
_ = plt.subplot(1,4,2), plt.imshow(grad_x, cmap=plt.cm.gray), plt.title('$G_x$')
_ = plt.subplot(1,4,3), plt.imshow(grad_y, cmap=plt.cm.gray), plt.title('$G_y$')
_ = plt.subplot(1,4,4), plt.imshow(mag, cmap=plt.cm.gray), plt.title('$G$')
```

Ex. 2.1

```
# Calculate the center of the image
query_image = aachen_day[450:1000, 200:900]
(h, w) = query_image.shape[:2]
(cX, cY) = (w//2, h//2)
# Rotate and scale query image
```

```python
M = cv2.getRotationMatrix2D((cX, cY), 45, 2.0)
db_image = cv2.warpAffine(query_image, M, (w, h))
# Display image
fig = plt.figure(figsize=(15, 20))
_ = plt.subplot(1,2,1), plt.imshow(query_image), plt.title('Query image')
_ = plt.subplot(1,2,2), plt.imshow(db_image), plt.title('Database image')
plt.show()

# Detect and compute keypoints and descriptors
query_keypoints, query_descriptors = sift.detectAndCompute(query_image, None)
db_keypoints, db_descriptors = sift.detectAndCompute(db_image, None)
print(f"\nNumber of keypoints (query image): {len(query_keypoints)}.")
print(f"Number of keypoints (database image): {len(db_keypoints)}.\n")
# # Distance matrix (rows correspond to query descriptors while columns to database descriptors)
# dist_matrix = np.zeros((len(query_descriptors), len(db_descriptors)))
# for i, desQ in enumerate(query_descriptors):
#   for j, desT in enumerate(db_descriptors):
#     dist_matrix[i, j] = np.linalg.norm(desQ-desT)
# More efficient implementation to compute pairwise distances
dist_matrix = sklearn.metrics.pairwise_distances(query_descriptors, db_descriptors)

# Find the closest database descriptor to each query descriptor
min_distances = dist_matrix.min(axis = 1)
db_descriptors_idx = np.argmin(dist_matrix, axis=1)

# Discard matches above the threshold
thrs = 25
query_descriptors_idx = np.where(min_distances <= thrs)[0]

# Select keypoints, descriptors and indices of retrieved matches
query_keypoints = [query_keypoints[idx] for idx in query_descriptors_idx]
query_descriptors = [query_descriptors[idx] for idx in query_descriptors_idx]
db_descriptors_idx = [db_descriptors_idx[idx] for idx in query_descriptors_idx]
db_keypoints = [db_keypoints[idx] for idx in db_descriptors_idx]
db_descriptors = [db_descriptors[idx] for idx in db_descriptors_idx]
min_distances = [min_distances[idx] for idx in query_descriptors_idx]

print(f"Number of matches: {len(min_distances)}")
# Print Euclidean distance, query descriptor index and database descriptor index of retrieved
matches
for k in range(len(min_distances)):
  print(f"Euclidean distance: {min_distances[k]:.2f}, Query Descriptor n.
{query_descriptors_idx[k]}, Database Descriptor n. {db_descriptors_idx[k]}")
```

Ex. 2.2 (Optional)

```
row_min = [500, 400, 300, 200, 0]
row_max = [700, 800, 900, 1000, 1063]
col_min = [700, 500, 300, 200, 0]
col_max = [900, 1100, 1200, 1400, 1600]

# List of execution times
exec_time = []
# List of sizes of cropped images
images_size = []

# Threshold matches
thrs = 25

for r_min, r_max, c_min, c_max in zip(row_min, row_max, col_min, col_max):
  start_time = time.time()

  # Calculate the center of the image
  query_image = aachen_day[r_min:r_max, c_min:c_max]
  images_size.append(query_image.shape[:2])
  (h, w) = query_image.shape[:2]
  (cX, cY) = (w//2, h//2)
  # Rotate and scale query image
  M = cv2.getRotationMatrix2D((cX, cY), 45, 2.0)
  db_image = cv2.warpAffine(query_image, M, (w, h))

  # Detect and compute keypoints and descriptors
  query_keypoints, query_descriptors = sift.detectAndCompute(query_image, None)
  db_keypoints, db_descriptors = sift.detectAndCompute(db_image, None)
  print(f"\nNumber of keypoints (query image): {len(query_keypoints)}.")
  print(f"Number of keypoints (database image): {len(db_keypoints)}.\n")

  # Pairwise distances
  dist_matrix = sklearn.metrics.pairwise_distances(query_descriptors, db_descriptors)

  # Find the closest database descriptor to each query descriptor
  min_distances = dist_matrix.min(axis = 1)
  db_descriptors_idx = np.argmin(dist_matrix, axis=1)
```

```
    # Discard matches above the threshold
    query_descriptors_idx = np.where(min_distances <= thrs)[0]

    # Select keypoints, descriptors and indices of retrieved matches
    query_keypoints = [query_keypoints[idx] for idx in query_descriptors_idx]
    query_descriptors = [query_descriptors[idx] for idx in query_descriptors_idx]
    db_descriptors_idx = [db_descriptors_idx[idx] for idx in query_descriptors_idx]
    db_keypoints = [db_keypoints[idx] for idx in db_descriptors_idx]
    db_descriptors = [db_descriptors[idx] for idx in db_descriptors_idx]
    min_distances = [min_distances[idx] for idx in query_descriptors_idx]

    # add exection time
    exec_time.append(time.time() - start_time)

_ = plt.figure(figsize=(6,6))
plt.plot(range(1, len(row_min)+1), exec_time, '--o'), plt.title('Execution time'), plt.grid('on')
# Integer x-axis
plt.locator_params(axis='x', integer=True, tight=True)
plt.xlabel('Experiment'), plt.ylabel('Time [s]')
for i in range(len(row_min)):
    plt.text(i, exec_time[i], f"      ({images_size[i][0]}, {images_size[i][1]})")
```

Ex. 3

```
# Calculate the center of the image
query_image = aachen_day[450:1000, 200:900]
(h, w) = query_image.shape[:2]
(cX, cY) = (w//2, h//2)
# Rotate and scale query image
M = cv2.getRotationMatrix2D((cX, cY), 45, 2.0)
db_image = cv2.warpAffine(query_image, M, (w, h))

# Detect and compute keypoints and descriptors
query_keypoints, query_descriptors = sift.detectAndCompute(query_image, None)
db_keypoints, db_descriptors = sift.detectAndCompute(db_image, None)

# Pairwise distances
dist_matrix = sklearn.metrics.pairwise_distances(query_descriptors, db_descriptors)

## K-NN matcher (K=2)
# For each query descriptor find the indices and distances
```

```
# of the 2 closest database descriptors.
k = 2
distances = np.partition(dist_matrix, kth=k, axis=1)[:, :k]
db_idx = np.argpartition(dist_matrix, kth=k, axis=1)[:, :k]

# Apply ratio test
min_dist = []
min_query_idx = []
min_db_idx = []
for k in range(distances.shape[0]):
   if distances[k][0] < 0.75 * distances[k][1]:
      min_dist.append(distances[k][0])
      min_query_idx.append(k)
      min_db_idx.append(db_idx[k][0])

print(f"Number of query keypoints: {len(query_keypoints)}")
print(f"Number of database keypoints: {len(db_keypoints)}")
print(f"Number of matches: {len(min_dist)}")

# Print first 10 matches
nMatches = 10
for k in range(nMatches):
   print(f"Euclidean distance: {min_dist[k]:.2f}, Query Descriptor: n. {min_query_idx[k]},
Database Descriptor: n.{min_db_idx[k]}")
```

Ex. 5

```
# Brute-force matcher
bf = cv2.BFMatcher_create(cv2.NORM_L2)

# Finds the best match for each descriptor from a query set.
matches = bf.match(query_descriptors, db_descriptors)

# Order matches
ordered_matches = sorted(matches, key = lambda m:m.distance)

# Discard matches above the threshold
thrs = 100

matched_des = [ordered_matches[idx] for idx in range(len(ordered_matches)) \
                  if ordered_matches[idx].distance <= thrs]
```

```
print(f"Number of matches: {len(matched_des)}")

print(f"Min distance: {matched_des[0].distance}, Max distance: {matched_des[-1].distance}.")


fig = plt.figure(figsize=(15,10))

draw_params = dict(matchColor=(0, 255, 0), singlePointColor=(255, 0, 0), flags=0)

out_matches = cv2.drawMatches(query_image, query_keypoints, database_image, db_keypoints,
matched_des[:50], None, **draw_params)

_ = plt.imshow(out_matches)
```